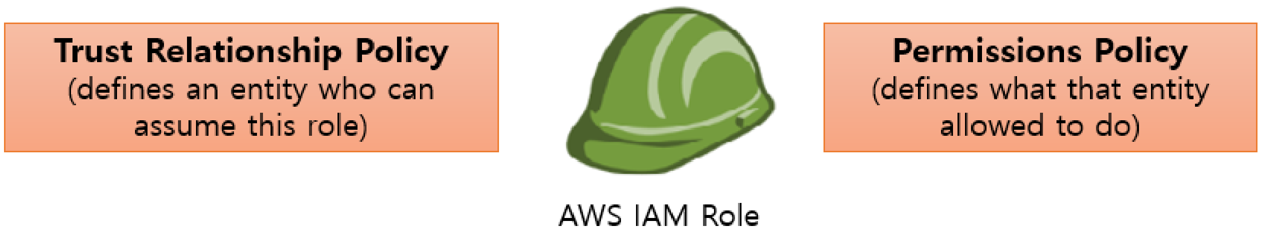
Programming AWS IAM using AWS python SDK boto3 — Part 3

Create an IAM role to delegate permissions to an AWS Service or an IAM user of another AWS account.

Manmohan Singh Bohara Follow
Apr 21, 2020 · 5 min read


Trust Relationship Policy
(defines an entity who can assume this role)
AWS IAM Role
Permissions Policy
(defines what that entity allowed to do)

In this article, we will see how to create an IAM role to delegate permissions to an AWS service and an IAM user of another AWS account. AWS IAM roles are very powerful. We should always try to use roles instead of access keys as much as possible.

1. Import boto3 and json library.

```
import json, boto3
```

2. Creating iam client representing AWS IAM services respectively.

```
iam_client = boto3.client('iam')
```

3. Defining trust relationship policy that defines which entity can assume this role. Following entities can assume an IAM role.

- An AWS service e.g. ec2, lambda.
- An IAM user of another AWS account.
- Third party identity provider. (**We will discuss this in detail in later sections of the tutorial**)

4. Below trust relationship policy can be used to assume this role by a particular service in the same account. In this case, the entity which can assume this role is AWS EC2 service.

Principal — defines the entity which can assume this role.

```
trust_relationship_policy_another_aws_service = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

5. Below trust relationship policy can be used to assume this role by an IAM user of another AWS account.

> *The account who defines trust relationship policy is termed as trusting account and the account who assumes this role to gain access is termed as trusted account. Don't forget to replace the iam user's arn of trusted account in below code.*

```
trust_relationship_policy_another_iam_user = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::
<PASTE_HERE_ACCOUNT_ID_OF_ANOTHER_AWS_ACCOUNT>:user/<USER_NAME>"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

6. Once trust relationship policy is defined, we can create role using *create_role()* method provided my boto3 iam client. *create_role()* method takes various parameters.

RoleName and *AssumeRolePolicyDocument* are mandatory parameters. *AssumeRolePolicyDocument* is a json string which takes above defined trust relationship policy as input.

Make sure to convert the policy into json string using *json.dumps()* method before passing it as parameter. We are also using two optional parameters *Description* as string and *Tags* as list of dict of key/value pairs.

```
try:
    create_role_res = iam_client.create_role(
        RoleName=role_name,

AssumeRolePolicyDocument=json.dumps(trust_relationship_policy_anothe
r_iam_user),
        Description='This is a test role',
        Tags=[
            {
                'Key': 'Owner',
                'Value': 'msb'
            }
        ]
    )
except ClientError as error:
    if error.response['Error']['Code'] == 'EntityAlreadyExists':
        return 'Role already exists... hence exiting from here'
    else:
        return 'Unexpected error occurred... Role could not be
created', error
```

7. Once the role is created, we can create an IAM policy for this role. We have already discussed this in previous section. The following policy provides full EC2 access to the entity which assumes above role.

```
policy_json = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "ec2:*"
        ],
        "Resource": "*"
    }]
}

policy_name = role_name + '_policy'
policy_arn = ''

try:
    policy_res = iam_client.create_policy(
        PolicyName=policy_name,
        PolicyDocument=json.dumps(policy_json)
    )
    policy_arn = policy_res['Policy']['Arn']
except ClientError as error:
    if error.response['Error']['Code'] == 'EntityAlreadyExists':
        print('Policy already exists... hence using the same 
policy')
        policy_arn = 'arn:aws:iam::' + account_id + ':policy/' + 
policy_name
    else:
        print('Unexpected error occurred... hence cleaning up', 
error)
        iam_client.delete_role(
            RoleName= role_name
        )
        return 'Role could not be created...', error
```

8. Now we will use *attach_role_policy()* method to attach above policy to our role.

```
try:
    policy_attach_res = iam_client.attach_role_policy(
        RoleName=role_name,
        PolicyArn=policy_arn
    )
except ClientError as error:
    print('Unexpected error occurred... hence cleaning up')
    iam_client.delete_role(
        RoleName= role_name
    )
    return 'Role could not be created...', error
```

Below is complete code to create an IAM role which can be assumed by an IAM user of trusted AWS account and have full access to EC2 resources in trusting account.

```
import json, boto3
from botocore.exceptions import ClientError


def lambda_handler(event, context):

    iam_client = boto3.client('iam')

    role_name = event['RoleName']
    account_id = event['AccountId']

    # Following trust relationship policy can be used to provide
access to assume this role by a particular IAM user from different
AWS acccount
    trust_relationship_policy_another_iam_user = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "AWS":
"arn:aws:iam::087851441689:user/<REPLACE_WITH_USER_NAME>"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }

    #Following trust relationship policy can be used to provide
access to assume this role by a particular AWS service in the same
account

    trust_relationship_policy_another_aws_service = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "ec2.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }

    try:
        create_role_res = iam_client.create_role(
            RoleName=role_name,

AssumeRolePolicyDocument=json.dumps(trust_relationship_policy_anothe
r_iam_user),
            Description='This is a test role',
            Tags=[
                {
                    'Key': 'Owner',
                    'Value': 'msb'
                }
            ]
        )
    except ClientError as error:
        if error.response['Error']['Code'] == 'EntityAlreadyExists':
            return 'Role already exists... hence exiting from here'
        else:
            return 'Unexpected error occurred... Role could not be
created', error

    policy_json = {
        "Version": "2012-10-17",
        "Statement": [{
            "Effect": "Allow",
            "Action": [
                "ec2:*"
            ],
            "Resource": "*"
        }]
    }

    policy_name = role_name + '_policy'
    policy_arn = ''

    try:
        policy_res = iam_client.create_policy(
            PolicyName=policy_name,
            PolicyDocument=json.dumps(policy_json)
        )
        policy_arn = policy_res['Policy']['Arn']
    except ClientError as error:
        if error.response['Error']['Code'] == 'EntityAlreadyExists':
            print('Policy already exists... hence using the same
policy')
            policy_arn = 'arn:aws:iam::' + account_id + ':policy/' +
policy_name
        else:
            print('Unexpected error occurred... hence cleaning up',
error)
            iam_client.delete_role(
                RoleName= role_name
            )
            return 'Role could not be created...', error

    try:
        policy_attach_res = iam_client.attach_role_policy(
            RoleName=role_name,
            PolicyArn=policy_arn
        )
    except ClientError as error:
        print('Unexpected error occurred... hence cleaning up')
        iam_client.delete_role(
            RoleName= role_name
        )
        return 'Role could not be created...', error

    return 'Role {0} successfully got created'.format(role_name)
```

> *I have tested this code on the AWS lambda console. However this should also work on the box where python 3.7, boto3 and aws cli is installed and aws cli is configured appropriately to access IAM.*

We need to provide sufficient permissions to the entity (e.g. Lambda) which would execute this code. I am using following policy to provide sufficient permissions to Lambda so that it can make IAM, SES and CloudWatch API calls.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:us-east-1:827178005985:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:827178005985:log-
group:/aws/lambda/createIAMRole:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:TagRole",
                "iam:AttachRolePolicy",
                "iam:CreatePolicy"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Source code is available here.

Conclusion

In this article, we learned how to create an IAM role and attach trust relationship policy, so that this role can be assumed by other resources or IAM user.

> ***If you find this article useful or if you see any improvements are needed, please do let me know below in the comment section.***

Programming AWS IAM using AWS python SDK boto3 — Part 2

Creating an IAM programmatic user and attaching a managed or custom policy and sending access and secret keys in email

medium.com

Programming AWS IAM using AWS python SDK boto3 — Part 4

Assume an IAM role in trusting AWS account from trusted AWS account and retrieve IAM group names attached to a given…

medium.com

Sign up for Geek Culture Hits

By Geek Culture

Subscribe to receive top 10 most read stories of Geek Culture — delivered straight into your inbox, once a week. Take a look.

Your email

Get this newsletter

Boto3 Aws Iam Iam Trust Relationship Iam Roles Automating Aws Iam